C# 프로그래밍

- 10 주차 (1강)

SEJONG UNIVERSITY

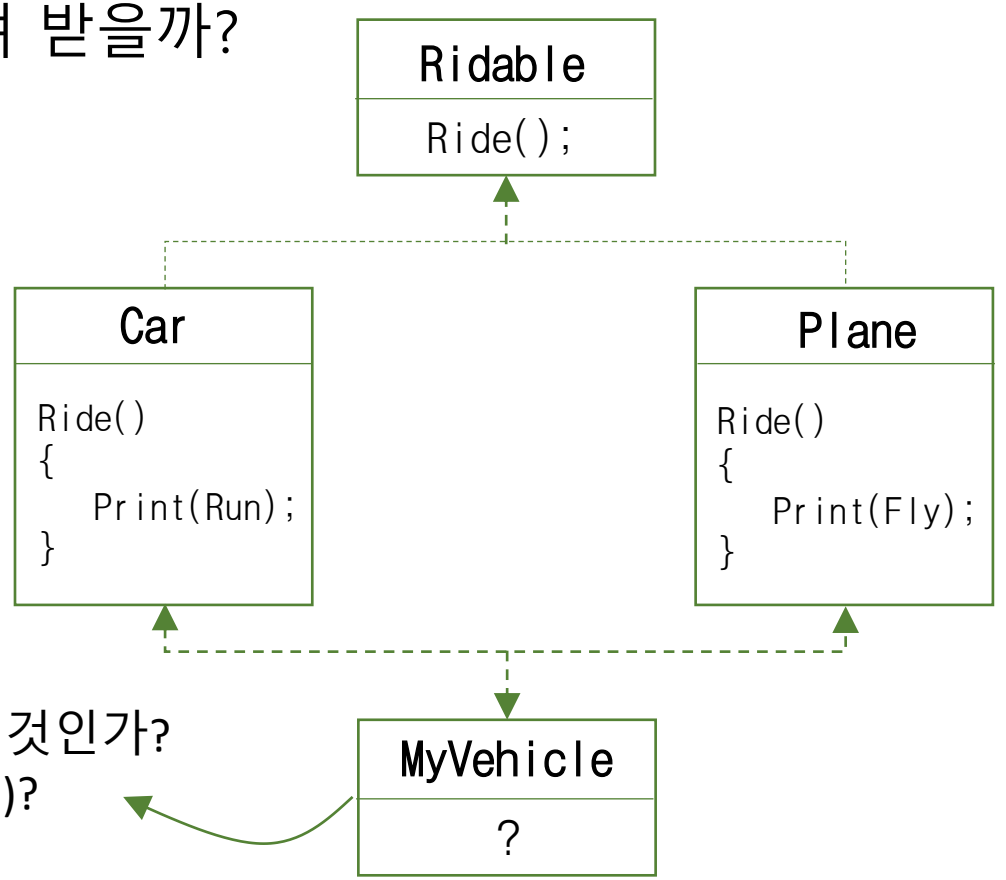
인터페이스

- 다중 상속
- 다형성
- 콜백
- 기본 구현 메소드

여러 개의 인터페이스, 한꺼번에 상속하기

- 클래스는 "죽음의 다이아몬드" 문제 때문에, 여러 클래스 한꺼번에 상속할 수 없음
  - ✓ 예제그림에서, 최초 클래스(Ridable)의 두 파생클래스(Car, Plane)가 존재하고,
  - ✓ 이 두 파생클래스를 다시 하나의 클래스(MyVehicle)가 상속 했을 때,
  - ✓ MyVehicle 클래스는 어느 Ride() 메소드를 물려 받을까?

- 인터페이스는 내용이 아닌 외형을 상속
  - ✓ "죽음의 다이아몬드" 문제 발생하지 않음



인터페이스의 다중 상속 지원

- 클래스와 달리 인터페이스는 다중 상속이 허용
  - ✓ 인터페이스의 메서드를 자식 클래스에서 구현할 때는 반드시 public 접근 제한자 명시
  - ✓ 예를 들어, Notebook 클래스에서는 Computer 클래스와 IMonitor, Ikeyboard 인터페이스를 모두 상속 받음

```
class Computer { }

interface IMonitor // 메서드 시그니처만을 포함하고 있는 인터페이스
{
    void TurnOn();
}

interface IKeyboard { } // 비어 있는 인터페이스 정의 가능

class Notebook : Computer, IMonitor, IKeyboard
{
    public void TurnOn() { } // 반드시 public 접근제한자 명시
}
```

```
static void Main(string[] args)
{
    Notebook notebook = new Notebook();
    notebook.TurnOn();
}
```

인터페이스의 다중 상속 지원

- 인터페이스의 메서드를 자식 클래스에서 구현할 때, 인터페이스 명을 직접 붙이는 경우 public 접근 제한자 생략 가능
  - ✓ 명시적으로 인터페이스의 멤버에 종속 시킨다는 표시
  - ✓ Notebook의 멤버로써 호출 불가능하고 인터페이스로 형변환하여 호출 가능

```
// ...

class Notebook : Computer, IMonitor, IKeyboard
{
    void IMonitor.TurnOn() { }  // 인터페이스 명을 직접 붙이는 경우
}
```

```
static void Main(string[] args)
{
    Notebook notebook = new Notebook();
    // notebook.TurnOn();   // IMonitor.TurnOn 메서드는 Notebook 인스턴스로 호출 불가능
                            // 컴파일 오류 발생

    IMonitor mon = notebook as IMonitor;
    mon.TurnOn();           // IMonitor 인터페이스로 형변환해서 호출
}
```

인터페이스 다중상속 예제 코드

```
interface IRunnable {
    void Run();
}

interface IFlyable {
    void Fly();
}

class FlyingCar : IRunnable, IFlyable
{
    public void Run()
    {
        Console.WriteLine("Run! Run!");
    }

    public void Fly()
    {
        Console.WriteLine("Fly Fly");
    }
}
```

```
static void Main(string[] args)
{
    FlyingCar car = new FlyingCar();
    car.Run();
    car.Fly();

    IRunnable runnable = car as IRunnable;
    runnable.Run();

    IFlyable flyable = car as IFlyable;
    flyable.Fly();
}
```

출력 결과:

```
Run! Run!
Fly Fly
Run! Run!
Fly Fly
```

인터페이스와 다형성

- 인터페이스의 메서드는 가상 메서드이기 때문에 다형성의 특징이 적용
  - ✓ C# 컴파일러는 인터페이스의 메서드를 가상메서드로 간주
  - ✓ Virtual/override 예약어 사용할 필요 없음 (사용시 컴파일 에러)

```
interface IDrawingObject
{
    void Draw();
}

class Line : IDrawingObject
{
    public void Draw() { Console.WriteLine("Line"); }
}

class Rectangle : IDrawingObject
{
    public void Draw() { Console.WriteLine("Rectangle"); }
}
```

```
static void Main(string[] args)
{
    IDrawingObject[] instances = new IDrawingObject[]
                                        { new Line(), new Rectangle() };

    foreach (IDrawingObject item in instances)
    {
        item.Draw(); // 인터페이스를 상속받는 객체의 Draw 메서드 호출
    }
}
```

출력 결과:

```
Line
Rectangle
```

인터페이스 자체로 의미 부여

- 비어 있는 인터페이스를 상속받는 것으로 의미 부여
  - ✓ 예를 들어, System.Object 클래스의 ToString을 재정의한 클래스만을 구분
  - ✓ 예제코드에서 ToString 메소드를 재정의한 클래스는 IObjectToString 인터페이스 상속

```
interface IObjectToString { } // ToString을 재정의한 클리스에만
                              // 사용될 빈 인터페이스 정의
class Computer { } // ToString을 재정의하지 않은 예제 타입

class Person : IObjectToString
{ // ToString을 재정의했다는 의미로 인터페이스 상속
    string name;
    public Person(string name)
    {
        this.name = name;
    }

    public override string ToString()
    {
        return "Person: " + this.name;
    }
}
```

```
private static void DisplayObject(object obj)
{
    if(obj is IObjectToString)
    { // 인터페이스 형변환 가능 체크
        Console.WriteLine(obj.ToString());
    }
}

static void Main(string[] args)
{
    DisplayObject(new Computer());
    DisplayObject(new Person("홍길동"));
}
```

출력 결과:

```
Person: 홍길동
```

인터페이스를 이용한 콜백

- 인터페이스의 메서드를 상속된 클래스에서 반드시 구현해야 한다는 점을 이용

```
interface ISource
{// 콜백용으로 사용될 메서드를 인터페이스로 분리
    int GetResult();
}

class Source : ISource
{
    public int GetResult() { return 10; }

    public void Test()
    {
        Target target = new Target();
        target.Do(this);
    }
}
```

```
class Target
{ // Source 타입이 아닌 ISource 인터페이스를 받음
    public void Do(ISource obj)
    { // 콜백 메서드 호출
        Console.WriteLine(obj.GetResult());
    }
}

class MainApp
{
    static void Main(string[] args)
    {
        Source src = new Source();
        src.Test();
    }
}
```

출력 결과:
10

인터페이스를 이용한 콜백 : Array.Sort

- Array.sort 메소드는 배열만 인자로 받으면 기본적으로 배열을 오름차순으로 정렬
- Array.sort 메소드에 IComparer 인터페이스 인자를 사용하는 경우,
  - ✓ IComparer 인터페이스의 Compare 메소드 구현에 따라 내림차순 정렬도 가능

C#에 정의 된 Array 클래스의 Sort 메소드:

```
public static void Sort(Array array); // 오름차순 정렬
public static void Sort(Array array, IComparer? comparer); // 오름차순 또는 내림차순 결정 가능
```

C#에 정의 된 IComparer 인터페이스:

```
namespace System.Collections
{
    // x가 y보다 크면 1, 같으면 0, 작다면 -1을 반환하는 것으로 약속된 메서드
    public interface IComparer
    {
        int Compare(object? x, object? y);
    }
}
```

Array.Sort 예제 코드

```
class IntegerCompare : IComparer // IComparer를 상속받는 타입 정의
{
    // IComparer 인터페이스의 Compare 메서드를 구현
    // 이 메서드는 Array.Sort 메서드 내에서 콜백으로 호출됨
    public int Compare (object x, object y)
    {
        int xValue = (int)x;
        int yValue = (int)y;

        if (xValue > yValue) return -1; // 내림차순 정렬이 되도록 -1을 반환
        else if (xValue == yValue) return 0;

        return 1;
    }
}
```

```
static void Main(string[] args)
{
    int[] intArray = new int[] { 1, 2, 3, 4, 5 };

    // Array.Sort(intArray); // 오름차순 정렬
    // IComparer를 상속받은 IntegerCompare 인스턴스 전달
    Array.Sort(intArray, new IntegerCompare()); // 내림차순 정렬
    foreach (int item in intArray)
    {
        Console.Write(item + ", ");
    }
}
```

출력 결과:

5, 4, 3, 2, 1,

인터페이스를 사용한 느슨한 결합

- 강력한 결합(tight coupling)
  - ✓ 클래스 간의 호출
  - ✓ 유연성이 떨어진다는 약점이 있음

```
class Computer
{
    public void TurnOn()
    {
        Console.WriteLine("Computer: TurnOn");
    }
}

class Switch
{
    public void PowerOn(Computer machine)
    { // Computer 타입을 직접 사용
        machine.TurnOn();
    }
}
```

Computer클래스를 Monitor 클래스로 교체 :

```
class Monitor
{
    public void TurnOn()
    {
        Console.WriteLine("Monitor: TurnOn");
    }
}


class Switch
{
    public void PowerOn(Monitor machine)
    { // Switch 클래스 코드 변경 필요
        machine.TurnOn();
    }
}
```

인터페이스를 사용한 느슨한 결합

- 느슨한 결합(loose coupling)
  - ✓ 예제코드에서 Monitor 클래스, Computer 클래스 모두 IPower 인터페이스를 상속 받기 때문에 Switch 클래스의 코드를 수정할 필요 없음

```
interface IPower
{
    void TurnOn();
}

class Monitor : IPower
{
    public void TurnOn()
    {
        Console.WriteLine("Monitor: TurnOn");
    }
}
```

```
class Computer : IPower
{
    public void TurnOn()
    {
        Console.WriteLine("Computer: TurnOn");
    }
}

class Switch
{
    public void PowerOn(IPower machine)
    {
        machine.TurnOn();
    }
}
```

인터페이스를 상속하는 인터페이스

- 기존 인터페이스에 새로운 기능을 추가한 인터페이스를 만들 때,
- 인터페이스를 수정하지 않고 인터페이스를 상속하는 인터페이스를 이용하는 이유
  - ✓ 상속하려는 인터페이스가 어셈블리로만 제공되는 경우
  - ✓ 이미 인터페이스를 상속하는 클래스들이 존재하는 경우

```
interface 파생 인터페이스 : 부모 인터페이스
{
    // ... 추가할 메소드 목록
}
```

```
interface ILogger
{ // 부모 인터페이스
    void WriteLog(string message);
}

interface IFormattableLogger : ILogger
{ // 파생 인터페이스 (부모 인터페이스의 모든 메소드를 그대로 물려받음)
    void WriteLog(string format, params Object[] args);
}
```

인터페이스를 상속하는 인터페이스 예제코드

```
interface Ilogger { // 부모 인터페이스
    void WriteLog(string message);
}

interface IFormattableLogger : ILogger { // 파생 인터페이스 (부모 인터페이스의 모든 메소드를 그대로 물려받음)
    void WriteLog(string format, params Object[] args);
}

class ConsoleLogger2 : IFormattableLogger
{
    public void WriteLog(string message)
    {
        Console.WriteLine("{0} {1}",
            DateTime.Now.ToLocalTime(), message);
    }

    public void WriteLog(string format, params Object[] args)
    {
        String message = String.Format(format, args);
        Console.WriteLine("{0} {1}",
            DateTime.Now.ToLocalTime(), message);
    }
}
```

```
static void Main(string[] args)
{
    IFormattableLogger logger = new ConsoleLogger2();
    logger.WriteLog("C# Programing.");
    logger.WriteLog("{0} + {1} = {2}", 1,1,2);
}
```

출력 결과:

```
2021-05-03 오후 6:58:30 C# Programing.
2021-05-03 오후 6:58:30 1 + 1 = 2
```

인터페이스의 기본 구현 메소드

- 인터페이스에 기본적인 구현체를 가지는 메소드 생성
  - ✓ 파생 클래스에서 해당 메소드를 구현하지 않아도 에러 발생하지 않음
  - ✓ 기본 구현메소드는 인터페이스 참조로 업캐스팅 했을 때만 사용 가능

```
interface ILogger
{
    void WriteLog(string message);
    void WriteError(string error) // 기본 구현 메소드
    {
        WriteLog(error);
    }
}

class ConsoleLogger : ILogger
{
    public void WriteLog(string message)
    {
        Console.WriteLine(message);
    }
}
```

```
static void Main(string[] args)
{
    ILogger logger = new ConsoleLogger();
    logger.WriteLog("System up");
    logger.WriteError("System Fail");

    ConsoleLogger clogger = new ConsoleLogger();
    clogger.WriteLog("System up");
    //clogger.WriteError("System Fail");  // 컴파일 에러
}
```

출력 결과:

```
System up
System Fail
System up
```

Thank you!